VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

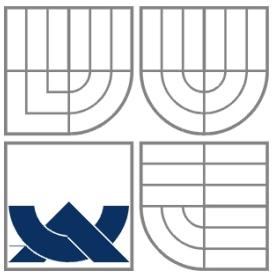
UMĚLÁ INTELIGENCE V MODERNÍCH POČÍTAČOVÝCH HRÁCH

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

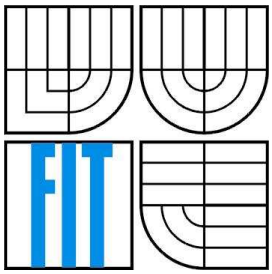
AUTOR PRÁCE
AUTHOR

JIŘÍ PAVLISKA

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

UMĚLÁ INTELIGENCE V MODERNÍCH POČÍTAČOVÝCH HRÁCH

ARTIFICIAL INTELLIGENCE IN MODERN COMPUTER GAMES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
JIŘÍ PAVLISKA

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. TOMÁŠ MIKOLOV

BRNO 2009

Abstrakt

Obsahem této bakalářské práce je návrh umělé inteligence pro moderní počítačovou hru. Snažím se seznámit čtenáře se samotným pojmem umělá inteligence. Uvádím popis prvků a technik používaných při tvorbě chování postav ovládaných počítačem. Zaměřuji se dále na prohledávání stavového prostoru a nalezení nejkratší cesty.

Abstract

The aim of this bachelor's thesis is to propose an artificial inteligence that could be used to create a modern computer game. My intention is to introduce the termin Artificial Inteligence to the reader. I include the description of elements and techniques used for determining the behavior of figures that are controlled by a computer. I am also focusing the abilities of the program to search the state space and to find the shortest possible way.

Klíčová slova

Umělá inteligence, nalezení optimální cesty, A*, prohledávání do šířky

Keywords

Artificial inteligence, pathfinding, A* , breadth-first search

Citace

Pavliska Jiří: Umělá inteligence v moderních počítačových hrách, bakalářská práce, Brno, FIT VUT v Brně, 2009

Umělá inteligence v moderních počítačových hrách

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Tomáše Mikolova. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jiří Pavliska
2.května 2009

Poděkování

Rád bych poděkoval svým rodičům, kteří mi umožnili studovat na této škole, a vedoucímu bakalářské práce Ing. Tomáši Mokolovovi za odbornou pomoc a poskytnutý čas.

Obsah

1 Úvod

Obsahem mé bakalářské práce je seznámit se s problematikou umělé inteligence v počítačových hrách. Jelikož se jedná o velmi rozsáhlý úkol zaměřil jsem se především na navrhnutí systému prohledávání a nalezení nejkratší cesty pro postavy ovládané počítačem a na navržení vhodných prostředků pro samotné rozhodování a inteligenci protihráče.

Je důležité si definovat pojem inteligence jak v počítačových hrách, tak i obecně. Tímto a dalšími termíny se budu zabývat v kapitole *Základní pojmy*.

Jelikož rozvoj umělé inteligence jde ruku v ruce s rostoucí kvalitou a rozvojem počítačových her, považuji za adekvátní zmínit se o jejich historii. Chci se zaměřit především na nástroje, které byly používány v nedávné minulosti. Neopomenu, samozřejmě, zmínku o úplných začátcích ani o technikách používaných v současné době. Vše je uvedeno spolu s návrhem chování počítačového protihráče v kapitole *Návrh inteligence počítačového protihráče*.

Pro úspěšnost na trhu a hratelnost všech her je důležité správně navrhnout a implementovat prohledávání krajiny resp. prostředí , ve kterém se protihráč pohybuje. K tomuto účelu se používalo a požívá mnoho přístupů a algoritmů. V kapitole *Nalezení nejkratší cesty* je vše podrobně rozebráno a popsáno. Zde uvádím popis podpůrného programu pro demonstraci hledání cesty v terénu a enginu, pomocí kterého je implementováno vykreslování a správný běh demonstračního programu pro prohledávání krajiny.

V kapitole *Závěr* zhodnocuji výsledky, které jsem zjistil při testování implementovaného prohledávacího algoritmu. Rozebírám také možnosti rozšíření a zapůsob integrace do herního projektu.

.

2 Základní pojmy

V této kapitole bych rád osvětlil význam nejčastěji používaných pojmů z oblasti umělé inteligence. Pro zachování logické posloupnosti nejprve rozeberu obecné názvy jako umělá inteligence, prohledávání prostoru, nalezení nejkratší cesty atp. Až poté rozeberu termíny, které jsou více specializované. V této kapitole také popisuji druhy použitých abstraktních datových typů.

2.1 Umělá inteligence

Obecně je pojem umělé inteligence (dále UI) definován jako schopnost počítače nebo stroje vykonávat činnosti, u kterých se běžně předpokládá, že k provedení daných úkolů vyžadují inteligenci.[1] Tato definice úzce souvisí s *Turingovým testem*, který určuje zda se jedná o umělou inteligenci či nikoliv.

V počítačových hrách je ovšem možné považovat za umělou inteligenci jakékoliv chování, které činí hru zajímavou a hratelnou. Protože cílem umělé inteligence v počítačových hrách není vytvořit dokonalé chování nepřátel, ale hlavním úkolem je pobavit veřejnost.

Vývoj a způsoby implementace UI jsou pečlivě popsány v následující kapitole. Proto jen pro orientaci uvádím, že prvním krokem bylo použití předdefinovaného chování, potom konečných automatů. Dále pak různých systémů využívající fuzzy logiku, neuronových sítí a jiných sofistikovaných algoritmů.

2.1.1 Turingův test

Jak už z názvu vyplývá, turingův test sestavil roku 1950 britský matematik a logik Alan Turing.

Úkolem testu je určit zda se počítačový program chová inteligentně. Toho se dosáhne tak, že do oddělených místností umístíme počítač, osobu, která bude odpovídat (osoba A), a osobu, která bude klást otázky(osoba B). Účastníci testu mohou mezi sebou komunikovat pomocí neutrálních prostředků např. tisk na papír. Osoba B klade otázky a ty jsou náhodně předávány buď osobě A, nebo počítači. Potom co je otázka zodpovězena je předána zpět osobě B. Pokud osoba B nezjistí, na které otázky odpovídal počítač, a na které člověk, můžeme považovat chování počítače za inteligentní.[2]

2.1.2 Polling

Do českého jazyka přeloženo jako *dotazování*. Je to způsob komunikace mezi několika entitami. Spočívá v neustálém dotazování jednoho prvku na stav nebo na provedené akce prvku druhého. V herním průmyslu se od této techniky opouští pro její neefektivitu. I v době, kdy se ve hře nic neděje, musí se jakýsi hlavní prvek ptát ostatních zda náhodou něco nestalo. [4]

2.1.3 Event-driven

V českém jazyce je toto spojení překládáno jako *událostmi řízené*. Na rozdíl od dotazování je komunikace mezi entitami zabezpečena pomocí posílání zpráv. Taková zpráva se vygeneruje pokaždé, když se ve hře dojde k nějaké události. Na první pohled je jasné, že způsob událostmi řízené komunikace je výhodnější pro vytížení procesoru. [4]

2.1.4 Konečný automat

Podrobnějším použitím konečných automatů v UI se budu zabývat v následujících kapitolách. Nyní pouze vysvětlím, co to konečné automaty jsou.

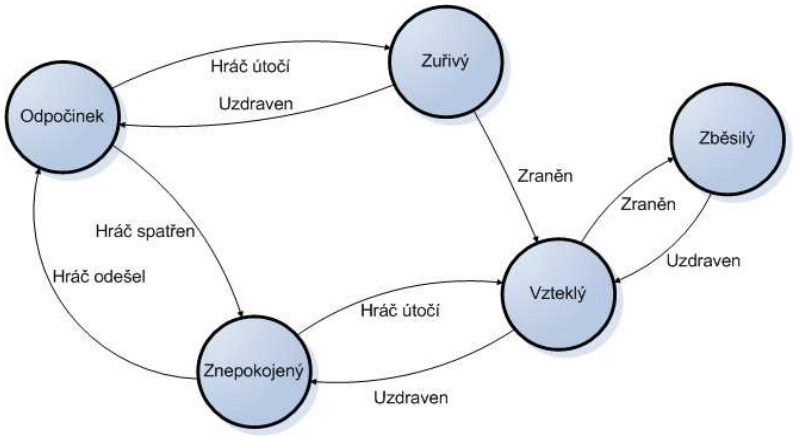
Konečný automat je pětice $M = (Q, \Sigma, R, s, F)$, kde

- Q je konečná množina stavů
- Σ je vstupní abeceda
- R je konečná množina pravidel ve tvaru $pa \rightarrow q$, kde p a $q \in Q$, $a \in \Sigma$
- s je počáteční stav a musí platit $s \in Q$
- F je množina koncových stavů a musí platit $F \subseteq Q$ [3]

Převedeno do lépe srozumitelné podoby. Konečný automat je obecný nástroj často využívaný v oblasti tvorby překladačů a v mnoha jiných odvětvích informatiky.

Ze vstupní pásky přečte informaci, která je obsažená v abecedě Σ. Podle stavu z množiny Q a definovaného přechodu v množině pravidel R se rozhodne, do kterého následujícího stavu z Q přejde.

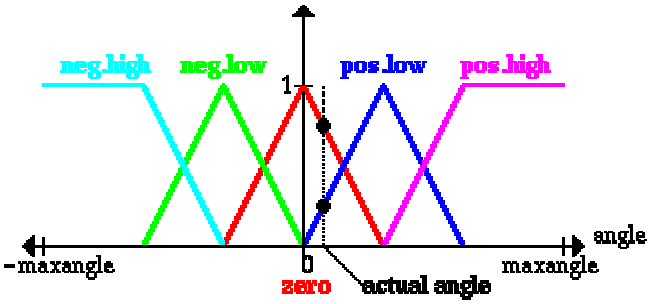
Konečné automaty je možné zapsat mnoha způsoby. Nejčastější reprezentace jsou pomocí grafů nebo tabulek.



Obr. 1 – Ukázka konečného automatu. Znázorňuje jednoduché chování monstra ve fiktivní Role Playing Game. Převzato z [4].

2.1.5 Fuzzy logika a fuzzy množiny

Fuzzy logika představuje rozšíření klasické boolovské logiky. Zatím co boolovská logika používá pro definování příslušnosti prvku k množině hodnoty 0 nebo 1 (např. buď je prvek členem množiny, nebo není). Fuzzy logika zavádí možnost částečného určení příslušnosti prvku do množiny. Pro tento účel byl zaveden pojem *stupeň příslušnosti*, který je dán *funkcí příslušnosti*.



Obr. 2 – Příklad rozdělení velikosti úhlu podle funkcí příslušnosti do jednotlivých fuzzy množin. Převzato z [5].

Jak vyplývá z obr. 2, funkce příslušnosti rozdělily množinu vyjadřující velikost úhlu na pět podmnožin (tj. záporný velký, záporný malý, nulový, kladný malý a kladný velký). Vyznačená velikost úhlu na grafu říká, že stupeň příslušnosti k nulovému úhlu je odhadem 0.75 a stupeň příslušnosti k kladnému malému úhlu je 0.25.

Tato schopnost vyjádřit přesně matematicky určitou nepřesnost nebo nejasnost v reálném světě vede k možnosti lépe popisovat a simulovat reálný svět. Systémy založené na fuzzy logice se používají v mnoha odvětvích průmyslu. Společně s neuronovými sítěmi jsou mocným nástrojem jak vytvořit umělou inteligenci schopnou napodobit co nejvěrněji chování a myšlení člověka.[6]

2.1.6 Počítačové učení

V počítačových hrách od devadesátých let minulého století vypukl hon za co nejrealističtější a nedůmyslnější UI. Má to příčinu v síle grafických adaptérů, které se dostávaní na trh. Proto se mohou vyvíjet hry, v kterých je dokonale zpracovaná grafika, a přesto zůstává dostatek zdrojů pro mocnější algoritmy řešící UI. Prvním takovýmto počinem je hra studia Lionhead Black & White. Ta využívá algoritmů vyžadující počítačové učení.

Počítačové (strojové) učení je schopnost systémů měnit své znalosti a vykonávat tak podobnou nebo stejnou činnost efektivněji. [8] Dělíme ji do tří okruhů:

- Učení bez učitele
- Učení s učitelem
- Posilované učení

Učení bez učitele

Učení bez učitele spočívá v nalezení podobností ve vektoru vstupních informací. Jiné informace většinou nebývají známé nebo dostupné. Při trénování se hledají tzv. shluky podobných vstupů (Clustering). V počítačových hrách se noc nevyužívá.[8]

Učení s učitelem

Učení s učitelem se provádí na tzv. trénovací množině. Pro tento druh učení je příznačné, že na každý vstup z trénovací množiny zná algoritmus správný výstup. [8]

Posilované učení

Od učení s učitelem se liší tím, že za každou provedenou akci dostává odměnu. Odměny mohou být různě velké, kladné i záporné. [8] Tento systém využívají počítačové hry.

Příklad posilovaného učení demonstruji na příkladu vojáka, který má za úkol zabít draka. Voják se rozhodne jít proti drakovi přímo. Drak si ho všimne a zraní ho dřív něž k drakovi voják dojde. Voják dostane zápornou odměnu na tento druh pohybu. Proto se rozhodne schovat se a postupovat k drakovi křovím. Drak si jej nevšimne a voják draka zabije. Dostane kladnou odměnu. Příště až bude voják útočit na draka bude se řídit podle toho, na který úkon získal nejvíce kladných odměn.

2.2 Pathfinding

Výraz pathfinding se do češtiny překládá jako slovní spojení *hledání cesty*. V herním světě je tím myšleno hledání nejkratší a optimální cesty v prostoru, ve kterém se pohybují jakékoliv entity, které

mají za úkol se dopravit z místa A do místa B. Hledání optimální cesty je ve většině her klíčové a je nedílnou součástí umělé inteligence.

2.2.1 Prohledávací algoritmy

Pro úplnost nejprve rozeberu význam slova algoritmus, i když by tento pojem měl být všem jasný. Jedná se o přesně definovanou konečnou posloupnost příkazů (kroků), jejichž prováděním pro každé přípustné vstupní hodnoty získáme po konečném počtu kroků odpovídající hodnoty výstupní. Algoritmus musí splňovat základní podmínky. Musí být konečný, obecný, deterministický a efektivní. [7]

Prohledávací algoritmy jsou nástroje, které dokáží zpracovat určitý graf. Ten reprezentuje daný prostor, ve kterém se objekt pohybuje. Výstupem prohledávacího algoritmu je trasa, po které se má objekt vydat. U nalezené trasy můžeme rozhodnout zda se jedná o *optimální cestu* či nikoliv. V počítačových hrách je v drtivé většině nutné optimální cestu nalézt. Je zapotřebí vybrat algoritmus, který optimální cestu dokáže najít, a ten použít do herního projektu. Existuje celá řada takovýchto algoritmů a některé jsou podrobněji popsány v následující kapitole.

2.2.2 Informované a neinformované prohledávací algoritmy

Nejprve uvedu a vysvětlím, co si představit pod pojmem *neinformované* prohledávací algoritmy. Jsou to algoritmy, které nemají žádné informace o poloze cílového bodu. Nemají ani možnost si tyto informace jakkoliv obstarat. Procházejí slepě celý prohledávaný prostor dokud nenaleznou cíl. Tyto algoritmy jsou často pomalé a hardwarově náročné, proto se pro implementaci nalezení nejkratší cesty v počítačových hrách nehodí. Patří mezi ně prohledávání do šířky (BFS), do hloubky (DFS), zpětného navrácení (Backtracking) atp. [8]

Algoritmy informované jsou více sofistikovanější. Mají informace o cílovém bodu, ke kterému je potřeba dojít. Mají i prostředky k ohodnocení cest vedoucí k cíli (např. algoritmus A*, který je popsán níže), proto jsou výrazně rychlejší a méně náročné na výkon počítače. Ty se už samozřejmě používají pro hledání cest v počítačových hrách. Zejména algoritmus A* byl a je velice používaným algoritmem.

2.2.3 Prohledávaný prostor – graf

Prohledávaný prostor nebo také stavový prostor je definován dvojicí (S,O). Kde S definuje množinu všech možných stavů $S=\{s_1,s_2,s_3,...\}$ a symbol O označuje množinu všech přechodů $O=\{o_1,o_2,...o_n\}$ umožňující měnit stavy S.

Nalezení cesty v takto definovaném prostoru je potom úloha, která je daná dvojicí (s_0,G). Kde s_0 označuje počáteční stav a musí platit $s_0 \in S$. G pak označuje množinu cílových stavů a platí

$G \subset S$. Samotné řešení nalezení cesty je posloupnost přechodů $s_1 = o_1(s_0)$, $s_2 = o_2(s_1) \ldots$ $s_n = o_n(s_{n-1})$ $s_n \in G$.[8]

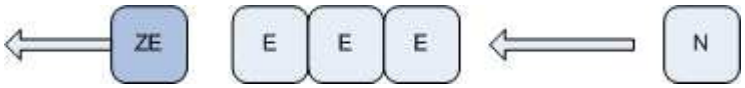
2.3 Použité abstraktní datové typy

Zde uvedu seznam abstraktních datových typů (ADT) použitých dále v práci. Měly by být obecně známé, ale považuji za adekvátní se o ADT zmínit alespoň ve zkratce.

Společnou vlastností ADT fronty a zásobníku je linearita. Z toho vyplývá, že jsou implementovány nad strukturami pole, jednosměrným seznamem nebo dvojsměrným seznamem.

2.3.1 Fronta

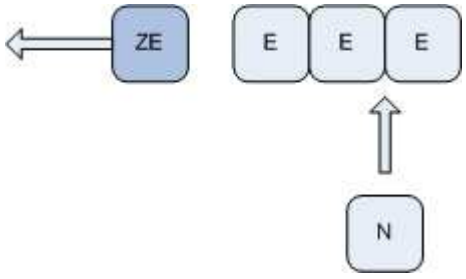
Je jedním ze základních datových typů. Fronta je dynamická, lineární a homogenní struktura. Často se frontě říká ADT typu FIFO (First In First Out). Do českého jazyka lze toto slovní spojení přeložit: „Kdo dřív přišel, ten dřív odchází“.[12] (viz. obr. 3)



Obr. 3 – Ukázka fronty. Element *ZE* je začátkem fronty a čeká na zpracování. Elementy *E* čekají ve frontě a nový prvek *N* se zařazuje na konec fronty.

2.3.2 Fronta s prioritou

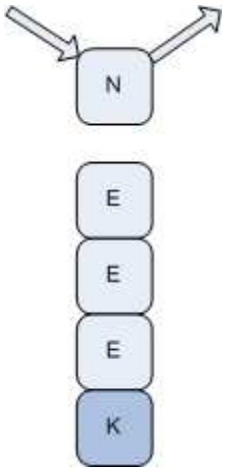
Fronta s prioritou je velmi podobná ADT fronta. Výběr prvku z fronty s prioritou je stejný. Vždy je vybírán prvek na začátku fronty. Ale vkládání prvku se mírně liší. Už se nově příchozí prvek nevkládá nakonec fronty, ale pomocí porovnávací funkce se určí, na kterou pozici bude nový prvek vložen. (viz. obr. 4)



Obr.4 – Ukázka fronty s prioritou. Element ZE je na začátku fronty a čeká na zpracování. Elementy E čekají ve frontě. Nový element N bude zařazen do fronty podle stanovené podmínky.

2.3.3 Zásobník

ADT zásobník je v jistém slova smyslu opakem fronty. Také se jedná o lineární, dynamický, homogenní typ, ale na rozdíl od fronty je strukturou typu LIFO (Last In First Out). Do jazyka českého přeloženo jako: „Kdo přijde poslední, půjde první pryč.“ Patří mezi nejvýznamnější ADT. (viz. obr.5) [12]



Obr.5 – Ukázka zásobníku. Prvek K je na konci zásobníku. Elementy E jsou uloženy v zásobníku. Element N je nový prvek zásobníku a bude přidán na jeho vrchol. A prvek jako první bude ze zásobníku vybrán.

3 Návrh inteligence počítačového protihráče

V této kapitole popisuji historii technik používaných k implementaci UI od samého počátku herního světa. A pokusím se uvést mnou navržený model rozhodování počítačového protihráče.

3.1 Historický vývoj UI

Při vzniku počítačových her, v dobách, kdy světu vládly ještě sálové počítače, a zrod personálních počítačů byl ještě daleko, vyšla v roce 1962 první počítačová hra Space War. Zde ještě nemůžeme mluvit o nějaké umělé inteligenci, protože hra byla určena jen pro dva hráče. Navíc výkon tehdejších počítačů neumožňoval výpočty v reálném čase spojené UI.

V sedmdesátých letech s nástupem personálních počítačů se výkon počítačů začal zvyšovat. To způsobilo vznik počítačem ovládaných nepřátel. Ovšem stále nemůžeme mluvit o zcela čisté UI. Zpočátku měli nepřátelé předdefinované chování, které se během hry neměnilo. Existovaly pouze seznamy jednoduchých úkonů, ze kterých se během hry vybíraly patřičné reakce.

Postupem doby se objevují náznaky nahodilosti. Ta je ale implementována předgenerovanou tabulkou náhodných čísel, pro snížení výpočetní náročnosti běhu hry.

Koncem osmdesátých a začátkem devadesátých let se začínají využívat techniky, které v některých hrách můžeme najít i dnes.

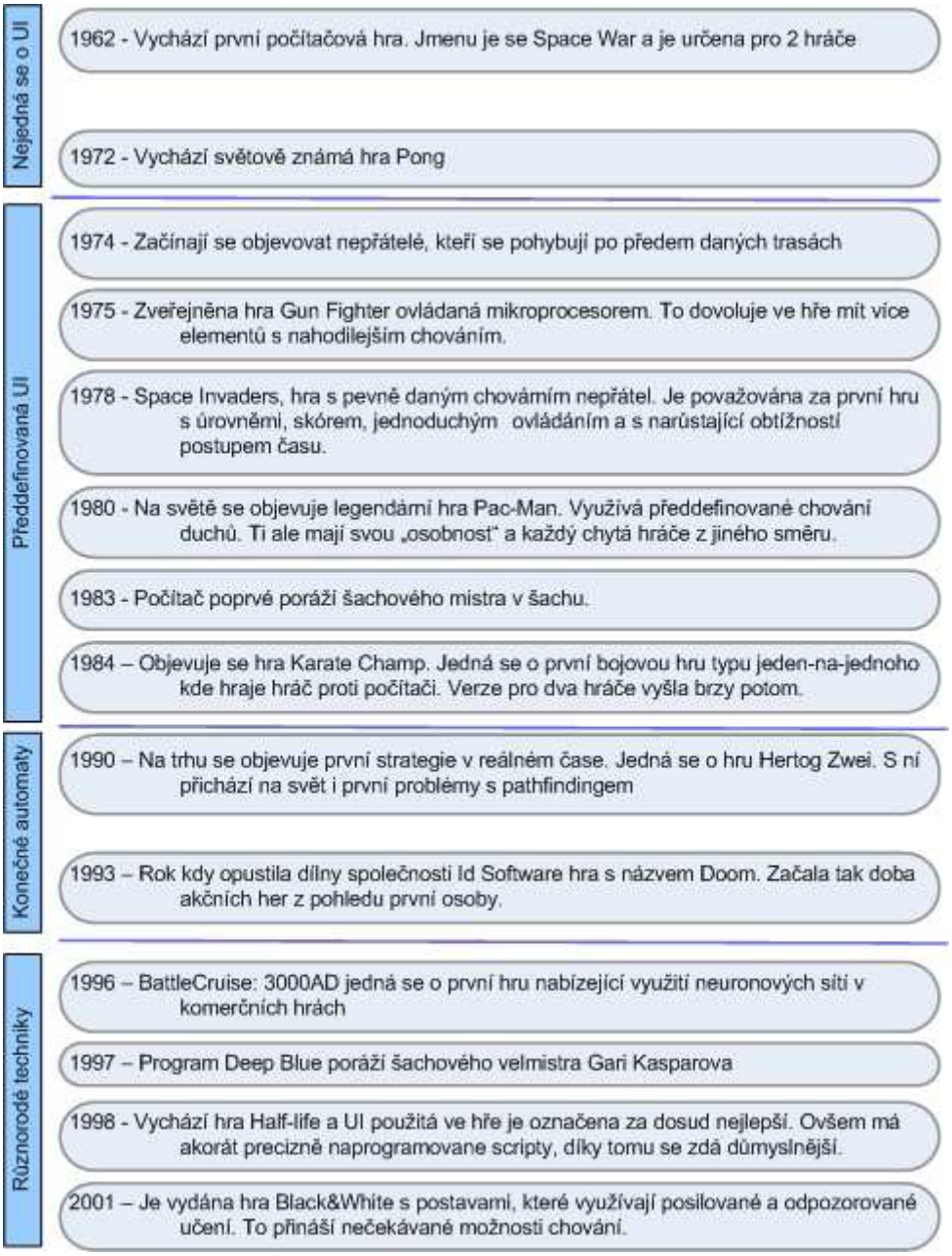
Z technického pohledu se jedná o využití konečných automatů s pozdějším výběrem akce podle naskriptovaných reakcí. K tomu účelu byly vytvořeny rychlé a relativně jednoduché skriptovací jazyky. Zářným příkladem je jazyk Lua. Využívaly se také obecné skriptovací jazyky např. Rubby nebo Python.

Z pohledu morálního začal počítač využívat jistý druh podvádění. V bojových hrách typu jeden-na-jednoho po stisknutí tlačítka pro kop do obličeje ještě než postava vedená člověkem vykopla, počítačem ovládaný protivník začal přehrávat animaci pro vykrytí. Ve strategiích bylo podvádění více markantní. Počítačem ovládané národy věděly o nalezištích surovin bez jakéhokoliv objevování terénu nebo měly nevyčerpatelné množství surovin.

I přes tyto výhody bylo chování počítače předvidatelné a stačilo počítači znemožnit přístup ke klíčovému bodu. Počítač neustále posílal jednotky k obsazeni patřičného místa a už se nevěnoval okolnímu světu.

Se stále narůstající výpočetní silou počítačů bylo možné takovéto problémy odstranit implementací sofistikovanějších algoritmů pro UI. Jednou z možností je místo klasických konečných automatů použít mocnější automaty počítající s fuzzy logikou nebo použití učících se algoritmů, z nichž nejpreferovanější jsou neuronové sítě. Za první hru, ve které byla použita efektivně neuronová síť považuji hru Black &White.

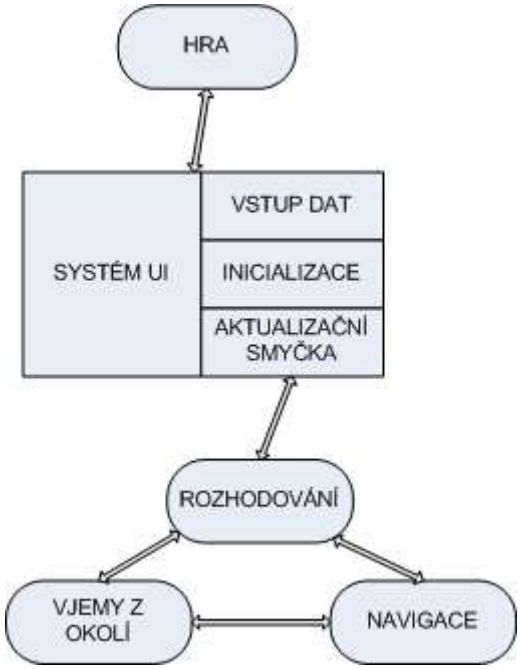
Na obr. 6 je uveden přehled vývoje UI v hrách.



Obr. 6 – Časový vývoj UI a milníky ve světě počítačových her [9]

3.2 Struktura UI

Jak už jsem naznačil UI v moderních hrách není jednotný celek. Je složena z několika podsystémů, které různě mezi sebou komunikují a předávají si informace. Na obr. 7 uvádím základní rozvržení systému UI.



Obr. 7 – Základní rozvržení struktury UI [9]

Základním prvkem celého systému je modul rozhodování. Podle toho, kterou hru vytváříme volíme různé implementační nástroje pro jeho tvorbu. V množství her to bývají konečné automaty někdy rozšířené o fuzzy logiku. V poslední době se začíná využívat vlastností neuronových sítí. Ovlivňujícím faktorem pro rozhodování je množství vjemů, které jsou pro něj dostupné. Navigace musí logicky přijímat informace z okolí, aby mohla správně vykonávat svou činnost a předává informace do rozhodovacího modulu.

Oblast popsaná jako systém UI obsahuje údaje, které mohou ovlivnit chování protihráče. Například úroveň zkušeností, schopnosti zacházet s předměty nebo charakteristické rysy postav. Obsahuje také nástroj pro umožnění komunikace mezi jednotlivými částmi modelu UI. Zahrnuje také mechanismus pro počáteční vstup dat, kde si např. vybíráme charakter, za který budeme hrát atd. Podle vybraných vlastností se celý systém musí inicializovat.

Samozřejmě se jedná o obecné rozložení a u různých her může vypadat rozvržení jinak.

3.3 Představení hry

Využité techniky pro vytvoření UI se liší žánr od žánru. Například ve válečných strategiích je zapotřebí dobře zajistit a nasimulovat týmové chování. Není příliš inteligentní, když lehcí a rychlí lukostřelci předběhnou před bitvou rytíře, kteří je mají chránit, a zaútočí na nepřátelskou jízdu. Ta je samozřejmě smete jedním protiútokem. Na druhou stranu, se tyto problémy v 3D akčních hrách typu Half-life nebo Quake III nemusí vůbec řešit. A sportovní simulátory, jako hry ze série Proevolution Soccer nebo jakékoliv hry od společnosti EA Sports, jsou kapitolou samy pro sebe.

Pro mou práci jsem si vybral žánr akční 3D hry z pohledu první osoby (jelikož se jedná o poněkud dlouhý název a obecně je zaběhlá zkratka FPS – First Person Shooter – dále budu raději používat zkratku). Příběh hry jsem neměl zapotřebí nějak výrazně zpracovávat, proto jsem vymyslel jednoduchou zápletku.

Je rok 2015 a americká armáda otevírá nový program pro výcvik bojových robotů. Vysadila jednotku nových adeptů na planetu podobnou Zemi. Roboti jsou poštváni proti sobě. Vítězem je ten nejlepší robot a odměnou je mu místo v první linii. V následující kapitole popíšu návrh implementace chování jednoho z robotů.

3.4 Návrh chování počítačového protihráče

Jako základní přístup ke komunikaci mezi objekty jsem zvolil událostmi řízenou koncepci. Princip je následující. Každý objekt ve hře může generovat zprávy. Pomocí nich oznamuje svou činnost. Tyto zprávy mohou být doručovány buď jednomu objektu ve hře nebo více objektům. Přijímání a vyřizování zpráv má v kompetenci směrovač. Zpravidla bývá implementován jako fronta s prioritou, do které se zprávy řadí podle času zpracování. Až nastane správný okamžik zpráva je vyzvednuta a předána příslušnému systému pro tvorbu rozhodnutí.

V mém případě je rozhodovací systém tvořen konečným automatem, který vybírá z množiny naskriptovaných reakcí tu odpovídající. Řešení UI využívající skriptované chování je velmi populární. A především u FPS her. Umožňuje hráčům naprogramovat si vlastní počítačové protihráče a hra se tak dál vyvíjí aniž by původní autor hry musel projevit větší iniciativu. Counter-strike a QuakeIII patří mezi hry využívající skriptované chování.

Mezi skriptovací jazyky používané pro podobné účely řadíme jazyk Lua. Jedná se o rychlý moderní skriptovací jazyk implementovaný v jazyce ANSI C. Můžeme se s ním setkat u počítačových her Baldur’s Gate nebo World of Warcraft. [15]

Velká výhoda u způsobu komunikace pomocí zasílání zpráv je snadné debugování. Jednoduše je možné ukládat zprávy do souboru a na první pohled je zřejmé, v jakém pořadí jednotlivé objekty reagovaly na určitou situaci.

3.5 Moderní používané techniky

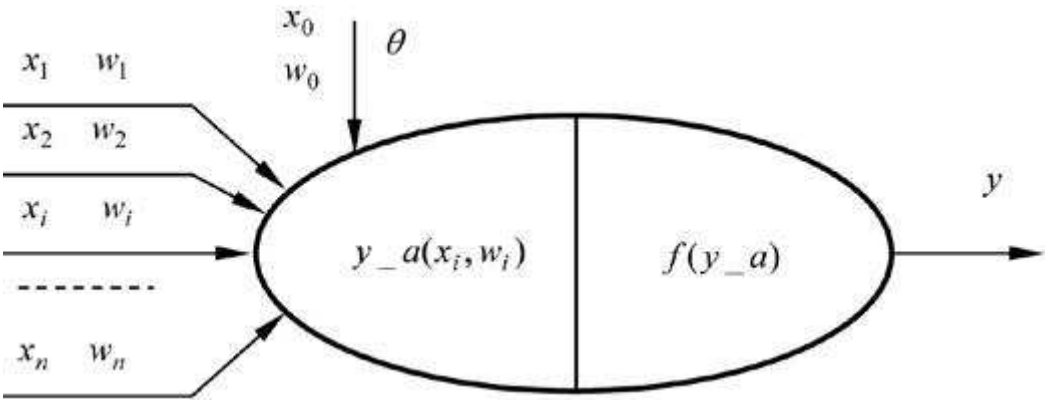
3.5.1 Neuronové sítě

Umělé neuronové sítě (dále NS) se snaží napodobit neuronové sítě biologické. Tedy vytváření umělých NS je z převážné části modelování jednak struktury a pak dále vlastní činnosti biologických NS.[10]

Je známo, že základní stavební jednotkou, jak strukturální, tak i funkční, v biologickém informačním modelu je nervová buňka – neuron. Analogicky je základním stavebním elementem umělé NS neuron, resp. jeho matematický model.

Obecně se můžeme setkat s velkým počtem modelů neuronu. Mohou se lišit topologií a matematickou funkcí, která definuje jejich chování. Doposud se nejvíce využívá tzv. formální model neuronu (obr. 8, na kterém vysvětlím jeho funkci a stavbu.

Mezi největší přednosti umělých neuronových sítí patří schopnost se učit, stejně jak tomu je u NS biologických.



Obr. 8– Model neuronu[11]

Nyní popíšu jednotlivé symboly z obr.3 a vysvětlím co znamenají[11]:

- $x_0, x_1, x_2, ..., x_n$ je vektor vstupních signálů do neuronu. Jedná se buď o výstupy jiných neuronů, nebo o vstupy z vnějšího prostředí. Obecně může vstup nabývat dvou typů forem a *kvantitativní* nebo *kvalitativní*.

  Mezi kvantitativní vstupy řadíme hodnoty získané např. měřením. Jedná se třeba o stav paliva v nádrži, rychlost vlaku atp. Jsou často vyjádřeny reálným číslem. Mezi kvantitativní vstupy se počítají také fuzzy proměnné.

  Kvalitativní vstup obvykle nabývá booleovských hodnot ano nebo ne. Popisu je tak situaci zda daný prvek v systému je nebo není obsažen.

- n je počet vstupů neuronu

- $w_0, w_1, w_2, ..., w_n$ jsou synaptické váhy. Jinak řečeno váhy spojení jednotlivých prvků NS. Nabývají často hodnot z množiny reálných čísel. Jelikož v NS je většina výstupních signálů neuronů vstupními prvky do jiných neuronů, synaptické váhy velmi výrazně ovlivňuji propustnost celé NS. Učení NS je z velké části zaměřen na dosažení správné propustnosti NS pomocí snižování nebo zvyšování vah jednotlivých spojů. Takto se snaží proces učení co nejvěrněji simulovat reálný systém.

- Θ je práh neuronu. V biologických NS je práh neuronu hodnota, kterou musí impuls minimálně mít, aby se mohl dále šířit. V umělých NS se práh neuronu používá často jen u neuronů přijímající informace z okolního světa. V neuronech uvnitř sítě se práh simuluje pomocí x_0 a w_0. Prahy neuronu se mohou v průběhu učení měnit.

- $y\_a(x_i, w_i)$ je agregační funkce neuronu. Jejím úkolem je určitým způsobem převést vstupní vektor $\vec{x}$ ovlivněný váhami $\vec{w}$ na skalární signál y_a. Ten je předložen na vstup aktivační funkce.

- $f(y\_a)$ je aktivační funkce neuronu. Převádí hodnotu vstupního potenciálu – výstup agregační funkce – na výstupní hodnotu neuronu. Funkce, která je použitá pro znázornění aktivační funkce, může být různá. Častým faktorem pro výběr funkce je rychlost jejího vyhodnocení.

- y je výstupní funkce neuronu. Dotváří jen okrajově výslednou hodnotu aktivační funkce. Simuluje přenos signálu do dalšího neuronu.

Nyní jsme již seznámeni s funkcí a složením neuronu, základního stavebního prvku NS. Je čas si říci něco přímo o umělé NS. Nejprve jen stručně k topologii. Neurony jsou řazeny do vrstev. Tyto vrstvy se rozlišují na tři typy podle polohy :

- vrstva vstupní. Obsahuje zdrojové uzly, které se starají o distribuci přijímaného podnětu z okolí. Zdrojové uzly nemají vlastnosti klasických neuronů, proto se tato vrstva vstupní nepočítá do výsledného počtu vrstev NS
- vrstva resp. vrstvy skryté
- vrstva výstupní. Ta přenáší výstupní signály z NS do okolí

Jak už jsem řekl výše, jednou z nejmocnějších vlastností neuronových sítí je schopnost se učit. Ve hrách to především znamená naučit se reagovat na situace, do kterých je počítačový protivník zatlačen hráčem. Proto se nejčastěji využívá posilovaného učení.

4 Nalezení optimální cesty

Jak jsem se už zmínil výše k prohledávání okolního terénu a nalezení optimální cesty se využívají různé algoritmy a různé reprezentace krajiny. Nejprve se zaměřím na popis jednotlivých grafů, které algoritmy zpracovávají a uvedu jejich výhody, nevýhody a typy her, pro které jsou výhodné.

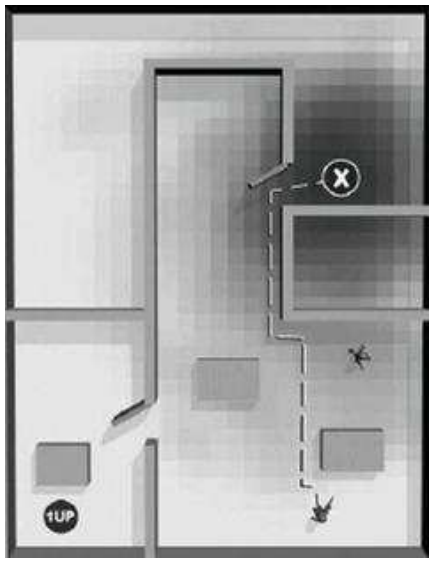
4.1 Způsoby navigace v terénu

4.1.1 Síťové rozložení – Grid base

Grid base rozložení je založeno na rozdělení oblasti pro pohyb hráčů na čtvercové nebo šestiúhelníkové plochy (obr. 9), které na sebe navazují. Jednotlivé plochy nesou informace s odkazem na své sousedy, informaci o možnosti projít danou plochou, jaký je na daném místě terén atp.

Nevýhodou u tohoto rozloženi je případná vysoká paměťová náročnost. U rozlehlých terénů s velikostí ploch nepoměrně malých může objem prohledávaných dat velmi výrazně zpomalit algoritmus, který se pokouší najít optimální cestu, natolik, že doba nalezení cesty bude větší než by bylo adekvátní.

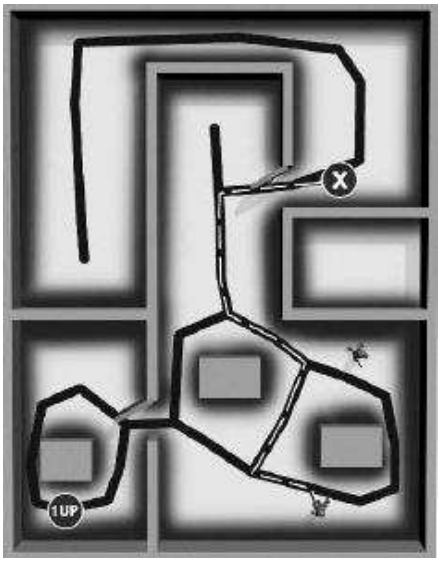
S tímto rozložením se můžeme setkat u tahových strategií, strategií v reálném čase a některých starších 2D hrách.



Obr.9– Ukázka síťového rozložení se čtvercovým tvarem ploch[9]

4.1.2 Navigace pomocí potenciálu

I zde je zapotřebí rozložit prostor do ploch, které se ale ohodnotí navíc *potenciálem*. A to takovým způsobem, že plochy blíže k překážce ohodnotíme vyšším potenciálem než plochy vzdálenější. Průchod takto rozvrstveným terénem připomíná pohyb v korytu. Pokud se postava začne pohybovat po plochách s vyšším potenciálem je usměrňována do prostorů s potenciálem nižším. V tomto druhu navigace vidím určitou jednoduchost a efektivnost, ale považuji ho za doplňkový způsob procházením terénu. Je dobré společně s ním implementovat sofistikovanější a mocnější způsob prohledávání terénu.



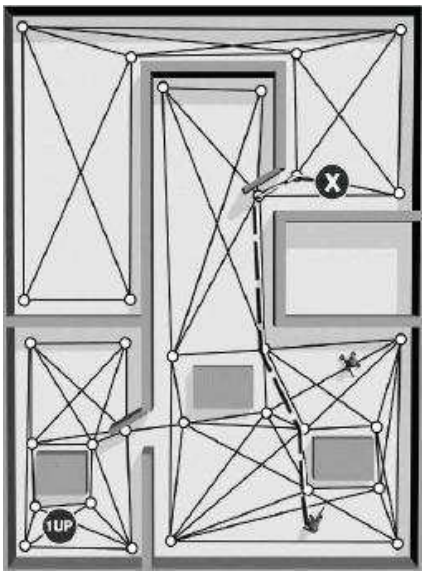
Obr.10 Navigace terénem pomocí potenciálu [9]

4.1.3 Navigace pomocí sítě uzlů

Patří mezi nejrozšířenější způsoby orientace v rozlehlých prostorech nebo ve 3D světě. Uzly jsou do terénu rozmístěny na strategická místa a spojeny cestami, po kterých může hráč přecházet do jednotlivých uzlů.

Nalezení optimální cesty je v této síti uzlů s pomocí správných prohledávacích algoritmů (např. A*) velice rychlé. Za rychlost se ale platí. Navigační síť nedokáže reagovat na dynamické podněty. Vhodným řešením je nenalézt cestu přesně k cíli, ale pouze k orientačnímu bodu nebo nejbližšímu uzlu a nadále pokračovat v jemnějším prohledávání.

Systémy zajišťující pohyb v prostoru založené na síti uzlů často využívají FPS hry. Ve hře Ureal Tournament dokonce bylo možné pomocí editoru vytvářet nové cesty a nové uzly. Tento systém je použit i ve hrách Half-life, Half-life 2, Quake III Arena a mnoho dalších.

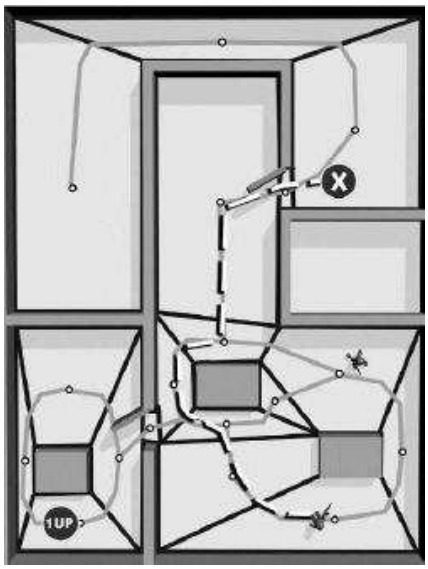


Obr.11 Ukázka navigace terénem pomocí sítě uzlů[9]

4.1.4 Navigace pomocí navigační sítě

Způsob prohledávání je zde stejný jako u sítě uzlů, ovšem v navigační síti jednotlivé uzly nereprezentují jen bod v prostoru, ale konvexní útvar – polygon. Z nich je sestavena celá mapa. Tento systém si zachovává všechny výhody orientace pomocí sítě uzlů. Polygony se prochází podobně jako uzly za použití stejných algoritmů. Nejčastěji se jedná o algoritmus A*.

Nejčastěji je tento přístup využíván v hrách typu her na hrdiny – RPG (Role Playing Game). Může se také používat při pohybu v budovách a uzavřených prostorách.



Obr.12 – Ukázka navigace terénem pomoci navigační sítě[9]

4.2 Prohledávací algoritmy

Základní definici pojmu jsem uvedl v druhé kapitole, proto si myslím že není zapotřebí delšího úvodu. V této sekci chci popsat a zhodnotit algoritmy používané pro hledání cesty grafem. Jako příklad neinformovaného prohledávání uvádím algoritmus prohledávání do šířky. Příklad informovaného prohledávání algoritmus A*. A jako ukázku sofistikovanějšího přístupu moderní algoritmus Ant colony.

4.2.1 Algoritmus prohledávání stavového prostoru do šířky

V zahraniční literatuře je často označován zkratkou BFS (Breadth First Search). BFS je základním a nejjednodušším prohledávacím algoritmem. Dokáže nalézt nejkratší cestu. Řadí se ovšem do skupiny neinformovaných prohledávacích algoritmů. Proto je výrazně pomalý a prohledává všechny stavy dokud nenalezne cílový uzel.

Algoritmus prohledávání do šířky je implementován pomocí fronty. Pro přehlednost ji nazvu fronta OPEN. Ve své zjednodušené verzi bez pomocného seznamu CLOSED je algoritmus ještě více výpočetně náročnější, protože nemá možnost si ukládat již jednou prozkoumané stavy. Může se tedy stát, že stejná místa algoritmus několikrát vyhodnotí a pokaždé zjistí, že daný stav není cílový. S použitím seznamu CLOSED odpadá tato starost. Každý procházený stav se nejprve zkontroluje, zda se nenachází v CLOSED. Když se stav v CLOSED nachází, je jasné, že stav byl algoritmem zpracován a nebyl to cílový uzel, proto bez dalších operací přejde na následující uzel z fronty OPEN.

Více demonstrativní a snazší na pochopení bude uvést algoritmus v pseudokódu. Uvádím variantu se seznamem CLOSED. Algoritmus jsem převzal a graficky upravil z [8]:

```
vytvoř frontu OPEN;
vytvoř seznam COLOSED;
ulož počáteční uzel do OPEN;
while ( OPEN není prázdná )
{
       vyber uzel z čela OPEN;
       ulož tento uzel do CLOSED;
       if( je uzel cílový )
       {
             vytvoř zpětně cestu;
             konec, cílový uzel byl nalezen;
       }
       ulož do OPEN následníky uzlu, pokud nejsou v CLOSED nebo OPEN
}
konec, cílový uzel nebyl nalezen;
```

Algoritmus prohledávání stavového prostoru do šířky se ve světě počítačových her se nepoužívá téměř vůbec. Je příliš pomalý a nemotorný, aby mohl být včleněn do UI her, kde je kladen důraz především na rychlost.

4.2.2 Algoritmus A*

Algoritmus A* je zástupcem informovaných prohledávacích algoritmů. Dokáže najít optimální cestu a je velice rychlý. Tyto vlastnosti záleží na principu ohodnocení uzlů. K expanzi a zpracování se vybírá jen ten nejlépe ohodnocený uzel. Ohodnocení uzlů je dáno funkcí:

$$f(n) = g(n) + h(n)$$

Kde $f(n)$ označuje výsledné ohodnocení uzlu, $g(n)$ představuje cenu od startovního uzlu po právě prohledávaný uzel a $h(n)$ znázorňuje *heuristickou* funkci, která určuje odhad ceny

k cílovému uzlu. Pro správnou funkčnost algoritmu A* je heuristická funkce klíčová. Využívá se mnoho funkcí pro odhad ceny cesty. Pokusím se představit alespoň několik z nich.

4.2.2.1 Heuristické funkce

Vzdálenost Manhattan

Heuristická funkce založená na manhattanovské vzdálenosti je velmi jednoduchá. Manhattan se této metodě určování vzdálenosti říká podle pravoúhlých ulic na Manhattonu. Když uvažujeme dvourozměrný prostor, pak funkce pro výpočet $h(n)$ má tvar:

$$h(n) = |x_n - x_{cíl}| + |y_n - y_{cíl}|$$

Kde poloha aktuálního bodu je $[x_n, y_n]$ a poloha cíle je dána $[x_{cíl}, y_{cíl}]$. Používá se pro odhad ceny cesty ve čtvercovém síťovém rozložení nejčastěji v dvourozměrném prostoru.

Eukleidovská vzdálenost

Zatímco vzdálenost Manhattan byla určena jen uzlům v pravoúhlé síti, eukleidovská vzdálenost je vhodná pro libovolnou síť reprezentující rozložení uzlů v terénu. Je použitelná pro dvourozměrný i trojrozměrný prostor. Určuje skutečnou vzdálenost mezi dvěma body. V 2D prostoru má tvar:

$$h(n) = \sqrt{(x_{cíl} - x_n)^2 + (y_{cíl} - y_n)^2}$$

V trojrozměrném prostoru pak rozšíříme výpočet o jednu souřadnici na:

$$h(n) = \sqrt{(x_{cíl} - x_n)^2 + (y_{cíl} - y_n)^2 + (z_{cíl} - z_n)^2}$$

Heuristika s možností vlivu terénu

Umožňuje započítat do výpočtu ceny uzlu i vliv terénu, po kterém se hráč pohybuje. Je to velmi užitečné a do her velmi vhodné řešení. Mnohdy není důležitá vzdálenost ale rychlost. Pokud půjde voják sice nejkratší cestou, ale zapadne v bažině, je to nepříjemné. Proto je dobré rozšířit rovnici odhadu ceny cesty o cenu uzlů. Získá potom tvar:

$$g(n) = g'(n) + c$$

$$f(n) = g(n) + h(n)$$

V první rovnici vypočítám $g(n)$ – celkovou cenu od startu – součtem ceny cesty od startu $g'(n)$ a cenou terénu c.[1]

4.2.2.2 Algoritmus

Jelikož jsem už popsal co je to heuristika a k čemu se používá, je načase popsat průběh algoritmu A*. V hodně zjednodušené verzi za určitých hodnot heuristické funkce se A* podobá algoritmu prohledávání do šířky. Což naznačuje i způsob implementace. I v A* je použita fronta, ale tentokrát fronta s prioritou. Pro přehlednost ji opět pojmenuji frontou OPEN. Stejně tak se zde vyskytuje i seznam prohledaných uzlů CLOSED. Průběh algoritmu je následující:

```
Vytvoř frontu s prioritou OPEN;
Vytvoř seznam CLOSED;
Inicializuj startovací uzel;
Vlož startovací uzel do OPEN;
while ( OPEN není prázdý )
{
	vyjmi první uzel z OPEN;
	if ( je to uzel cílový )
	{
		vytvoř zpětně cestu;
		konec, cílový uzel byl nalezen;
	}
	else
	{
		foreach( následující uzel z vyjmutého uzlu z OPEN)
		{
			vypočti novou cenu uzlu;
			if( následující uzel je v OPEN nebo CLOSED
				a je-li nová cena uzlu větší )
			{
				pokračuj v dalším následníkovi;
			}
			else
			{
				zpracuj následující uzel;
				if( je následující uzel v CLOSED )
				{
					odstraň jej s CLOSED;
				}
				if( je následující uzel v OPEN )
				{
					aktualizuj jej;
				}
				else
```

```
                        {
                              vlož následující uzel do OPEN;
                        }
                  }
            }
      }
      vlož uzel do CLOSED;
}
konec, cílový uzel nebyl nalezen;
```

[4]

Rychlost algoritmu je menší než u BFS. Je to dáno použitím heuristické funkce.Ta rozhoduje, který prvek je ohodnocen jako nejvýhodnější a ten je vybrán pro další zpracování. Musíme ovšem dávat pozor jak složitou heuristickou funkci použijeme. Pokud použijeme příliš složitou heuristiku, může algoritmus výrazně zpomalit. Proto je výhodnější někdy použít rychlou a ne příliš přesnou heuristickou funkci, která rozgeneruje více uzlů, než pomalou heuristiku s menším počtem propočítávaných uzlů.

5 Implementace

Vytvořil jsem systém navigující hráče v 3D terénu. Lze použít pro pohyb a nalezení optimální cesty jak pro postavy ovládané počítačem, tak i pro lidského hráče. Vše je vytvořeno nad obecným herním enginem nazvaným Wicked engine, který vzniknul pro účely této bakalářské práce (je obsažen v příloze 3). To zajišťuje teoretickou možnost zobrazování jakéhokoliv terénu podle výškové mapy s použitím adekvátních textur.

Jelikož jsem si zvolil při návrhu FPS hru, pohyb a navigace je řešený pomocí sítě uzlů. V herním projektu pro demonstraci by bylo z pohledu uživatele obtížné nakonfigurovat síť uzlů. Rozhodl jsem se proto vytvořit podpůrný program, který obstará pro demonstraci všechny potřebné vstupní parametry. Obsluha je popsána v souboru readme.txt, který se nachází v příloze 3 bakalářské práce. Vysvětlení činnosti podpůrného programu je v následující kapitole.

5.1 Aplikace Node manager

Podpůrný program jsem nazval Node manager. Je implementovaný v jazyce C#. Jak bylo zmíněno v předešlé kapitole zajišťuje správné vygenerování uzlů pro pohyb v terénu. Generuje cesty mezi těmito uzly a shromažďuje další potřebné informace o výškové mapě, o podobě demonstrovaného

prostředí a algoritmu použitého pro prohledávání sítě uzlů. Výšková mapa je načítána do programu ve formátu .raw.

5.1.1 Generování uzlů

Není přípustné, aby mohly být uzly vygenerované i v relativně nedostupných místech. Proto je v aplikaci Node manager zahrnuta možnost nastavit si výšku hladiny vody a maximální výšku pro pohyb na horách.

Výška hladiny vody reprezentuje nejnižší možnou úroveň, kde budou generovány uzly. V demonstračním programu bude do této úrovně generována voda. Ta bude tvořit přírodní překážku v pohybu po terénu.

Maximální výška pohybu v horách určuje největší možnou úroveň pro generování uzlů. A tvoří tak druhou přírodní překážku a omezení v pohybu po terénu.

5.1.2 Generování cest

Pro pohyb po krajině jen vygenerované uzly nestačí. Je zapotřebí definovat, kam je možné se z jednotlivých uzlů dostat. K tomu slouží automatické generování cest. V Node manageru je spuštěno ihned po vygenerování uzlů.

Podobně jako vygenerované uzly, ani cesty nemůžeme generovat všude. Cesty se generují pouze mezi nejbližšími uzly. A to jen při splnění podmínky:

$$\Delta H < povolenáHranice$$

$$\Delta H = \left| barvaPixelu - barvaNásledujícíhoPixelu \right|$$

Barva pixelu ve výškové mapě definuje výšku daného bodu ve výsledném zobrazovaném terénu.

5.1.3 Uživatelské vstupy

Nechat veškerou práci na počítači bez možnosti korekce by nebylo optimální řešení. Z toho důvodu je v Node manageru implementována možnost vytvářet uzly i cesty ručně a případně tak doplnit rozvržení sítě.

Pro správnou funkčnost obou aplikací je důležité vybrat výškovou mapu ve formátu raw. Zadat správně její velikost, zadat velikost skutečného terénu, vybrat prohledávací algoritmus a nastavit omezení pro generování uzlů. Všechny tyto údaje je zapotřebí uložit do souboru, podle kterého bude vykreslovat krajinu demonstrační program. Soubor se ukládá v mnou navrženém formátu .nod (ukázka v příloze 1).

Obr.13 – Ukázka aplikace Node Manager s načtenou výškovou mapou terénu

5.2 Demonstrační program

Demonstrační program je implementovaný v jazyce C++. Jazyk C++ jsem zvolil díky objektově orientovanému přístupu a jeho rychlosti. O vykreslování scény se stará univerzální herní jádro Wicked engine, které bylo vyvinuto pro demonstraci bakalářské práce i případné použití v jiných hrách. Využívá rozhraní directX 9, jehož instalační soubor je v příloze 3.

.

5.2.1 Načítání dat a vykreslování terénu

O načítání vstupních dat od uživatele (tzn. ze souboru vytvořeného aplikací Node Manager) se stará třída cFileManager. Pomocí metod definovaných nad objekty typu string rozloží soubor na patřičné části a ty posléze uloží do struktur vytvořených pro prohledávání stavového prostoru.

Vykreslování krajiny a objektů je v kompetenci herního enginu – Wicked engine. Ten je vytvořen tak, aby mohl být použitelný v různých herních projektech. Jeho zdrojové soubory jsou v příloze bakalářské práce (příloha 3).

Terén se vygeneruje podle výškové mapy. Na něj se nastaví patřičná textura. Textura se mapuje kolmo shora na terén. Prostorové objekty se do hry načítají také pomocí enginu. Umí načítat jakékoliv objekty vytvořené v 3DstudiuMax s koncovkou 3ds.

5.2.2 Algoritmus A* a BFS

Oba dva algoritmy jsou implementovány a optimalizovány pro rychlost vyhledávání. Pro přístup k položkám obsažených ve vektoru uzlů používám zásadně indexovou logiku. Po testování byla rychlejší než průchod iterátorem.

Stejně tak při vyhodnocování uzlů v algoritmu jsou co nejvíce využívány operace s indexy než s celými objekty.

V algoritmu A* byla použita rychlá a efektivní heuristická funkce pro výpočet vzdálenosti do cíle pomocí eukleidovské vzdálenosti. Výsledky a srovnání rychlosti algoritmů jsou uvedeny v následující kapitole.

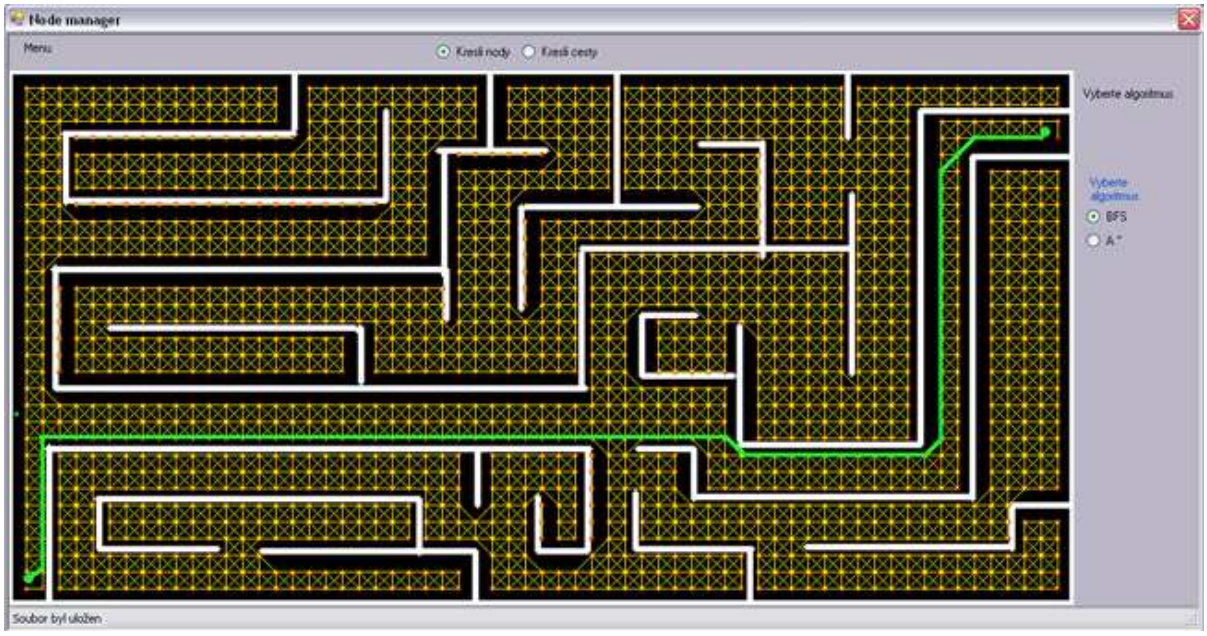
5.2.3 Pohyb objektu po terénu

Pohyb po terénu zprostředkovává také engine hry. Díky navrženému rozhraní v projektu wicked engine může uživatel pohybovat s kterýmkoliv objektem ve scéně včetně kamery. Engine také podporuje dědičnou strukturu objektů. V praxi to znamená např. snadnou realizaci pohledu z třetí osoby (kamera se nastaví jako potomek objektu, nad kterým má být umístěna).

V mém případě je pohyb po terénu řešen pomocí průsečíku středu obrazovky s terénem. Auto představující pohyblivý objekt vypočítá z aktuální pozice nejbližší dostupný uzel pro nalezení nejkratší cesty. Dopraví se k němu. Pomocí příslušného prohledávacího algoritmu zjistí nejkratší cestu. Z cíle se následně vydá nejkratší cestou k místu, na které bylo kliknuto.

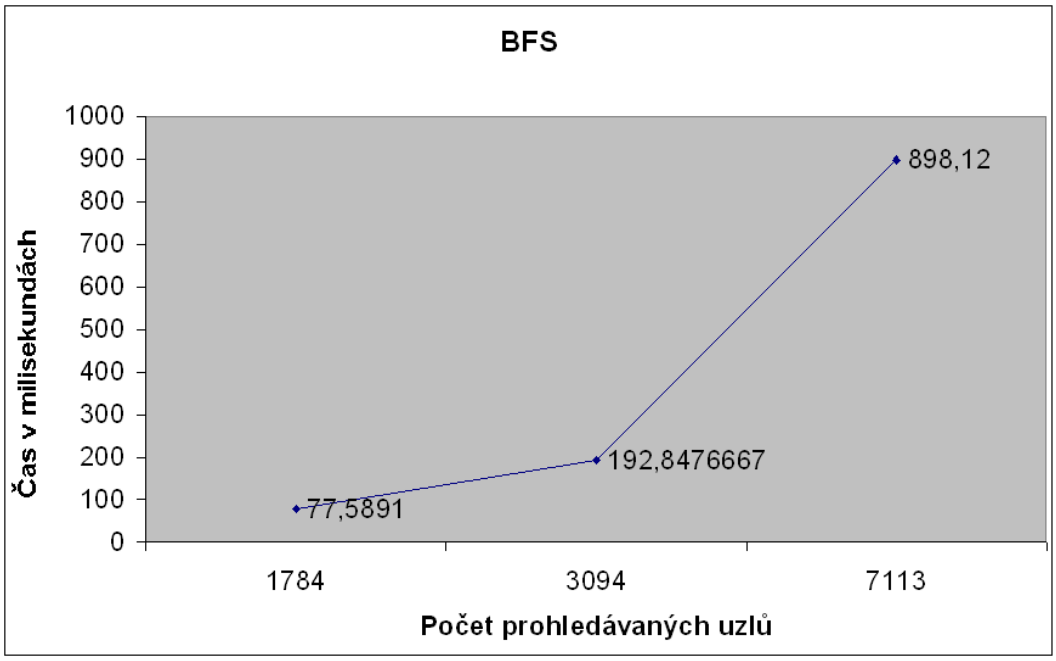
6 Zhodnocení výsledků

V demonstračním programu jsem se zaměřil na hodnocení rychlosti jednotlivých prohledávacích algoritmů. Vytvořil jsem testovací terén znázorňující bludiště (obr. 14).

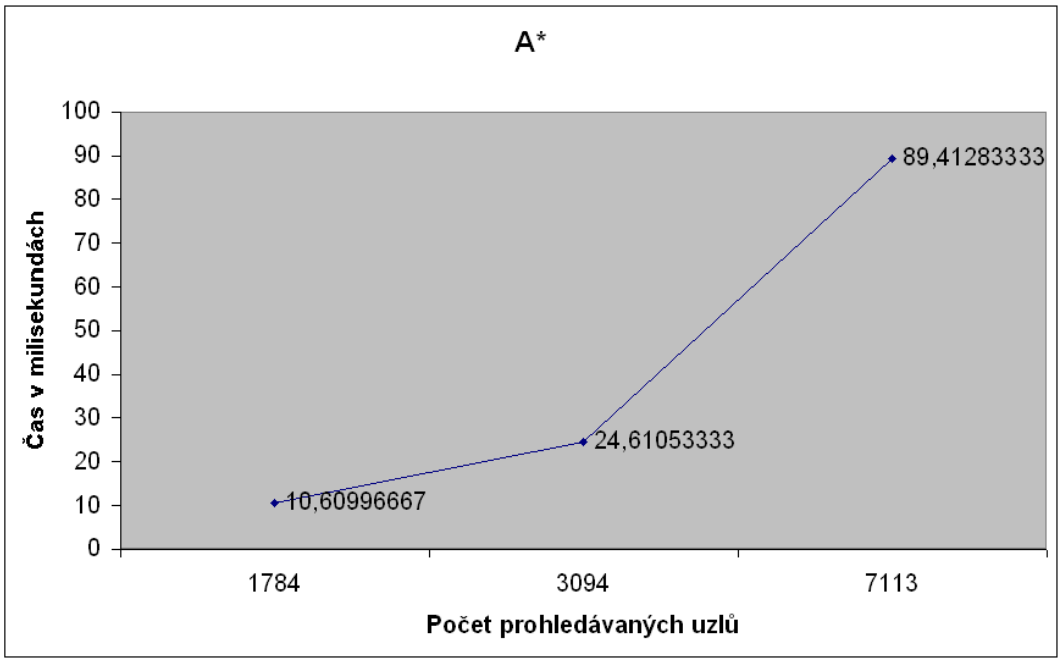


Obr.14 – Bludiště se sítí uzlů a cest. Zelenou barvou je vyznačena cesta ze startu do cíle.

Testoval jsem rychlost nalezení zeleně vyznačené cesty na algoritmech prohledávání do šířky a algoritmu A* na různém počtu vygenerovaných uzlů. Výsledky jsou uvedeny v následujících grafech. (graf 1 – metoda BFS a graf 2 – metoda A*).



Graf 1 – Průměrné časy prohledáváni stavového prostoru algoritmem BFS



Graf 2 – Průměrné časy prohledávání stavového prostoru algoritmem A*.

Jak je z grafů zřejmé, algoritmus prohledávání do šířky je na daném počtu uzlů řádově 10x pomalejší. Proto se ve hrách příliš nepoužívá. Snad jen na prohledávání velmi malých území nebo sítí s malým počtem uzlů.

Časy dosažené algoritmem A* prokazují proč právě tento algoritmus je většině her už dlouhou dobu používán. Pro případné urychlení prohledávání je možné omezit prohledávání prostoru do dálky. Určí se maximální vzdálenost, pro kterou je možné cestu najít, v jedné aktualizační smyčce. Tím směrem se auto vydá a při následujícím cyklu hry se dopočítává zbytek hledané cesty. Tento způsob nevytíží procesor na příliš velkou dobu a ve hře může být zpracováno více operací.

Aplikace byla testována na počítači s procesorem Intel Celeron M 1.5GHz s velikostí operační paměti 512 MB a grafickým adaptérem Ati Radeon 200X se sdílenou pamětí 128MB.

7 Závěr

Bakalářská práce se zabývá návrhem chování počítačového protihráče se zaměřením na nalezení optimální cesty terénem. Vybral jsem si toto téma, protože jsem chtěl prohloubit své znalosti v oblasti vývoje a tvorby moderních počítačových her.

Za účelem demonstrace hry byl vytvořen engine pro obecné použití. Na tomto jádru celý demonstrační program pracuje. Pro zpracování vstupů a vygenerování navigační sítě byla vytvořena aplikace Node Manager. V demonstračním programu je předváděna především funkčnost a rychlost prohledávacích algoritmů.

Modul pro prohledávání stavového prostoru je možné snadno zakomponovat i do jiných herních projektů. Třída zajišťující pathfinding je tvořená jako singelton, tím pádem s využitím stejných zdrojů hledá nejkratší cestu pro všechny objekty pohybující se po navigační síti.

Možné rozšíření projektu je začlenění navrženého systému chování z kapitoly číslo 3. Další možnost rozšíření vidím v implementaci neuronové sítě pro tvorbu rozhodování postav ovládaných počítačem.

Literatura

[1] Bourg, D. M.; Seemann, G.: *AI for Game Developers*. O'Reilly, 2004, ISBN 0-596-00555-5

[2] *Turingův test*. [online], [cit. 2009-05-04]. URL <http://cs.wikipedia.org/wiki/Turing%C5%AFv_test>

[3] Meduna A.; Lukáš R.: *Studijní opora IFJ Formální jazyky a překladače*. 2006, Brno

[4] DeLoura M.: *Game Programming Gems*. Charles River Media, 2000, ISBN 1-58450-049-2

[5] Bauer P.; Nouak S.; Winkler R.: *A brief course in Fuzzy Logic and Fuzzy Control*. [online], [cit. 2009-05-05], URL <http://www.esru.strath.ac.uk/Reference/concepts/fuzzy/fuzzy_ctrl.htm>

[6] Bauer P.; Nouak S.; Winkler R.: *A brief course in Fuzzy Logic and Fuzzy Control*. [online], [cit. 2009-05-05], URL <http://www.esru.strath.ac.uk/Reference/concepts/fuzzy/fuzzy.htm>

[7] Kreslíková J: *Základy programování – úvod do algoritmizace*. 2006, Brno

[8] Zbořil F.; Zbořil ml. F.: *Základy umělé inteligence IZU studijní opora*. 2006, Brno

[9] Schwab B.: *AI Game Engine Programming.* Charles River Media, 2004, ISBN 1-58450-344-0

[10] Drábek O.; Seidl P.; Taufer I.: *Umělé neuronové sítě – Základy teorie a aplikace (2).* 2005, [online], [cit. 2009-05-08], URL <http://www.chemagazin.cz/Texty/CHXV_6_cl2.pdf>

[11] Drábek O.; Seidl P.; Taufer I.: *Umělé neuronové sítě – Základy teorie a aplikace (3).* 2006, [online], [cit. 2009-05-08], URL <http://www.chemagazin.cz/Texty/CHXVI_1_cl3.pdf>

[12] Honzík J. M.: *Algoritmy IAL Studijní opora*. 2007, Brno

[13] Patel A.: *Heuristics*. 2008, [online], [cit. 2009-05-10], ULR <http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html>

[14] *DirectX*. [online], [cit. 2009-05-12], URL <http://msdn.microsoft.com/cs-cz/library/aa139818(en-us).aspx>

[15] *Lablua – Programming Language Research*.2006, [online], [cit 2009-05-19], URL <http://www.lua.inf.puc-rio.br/luanet/>

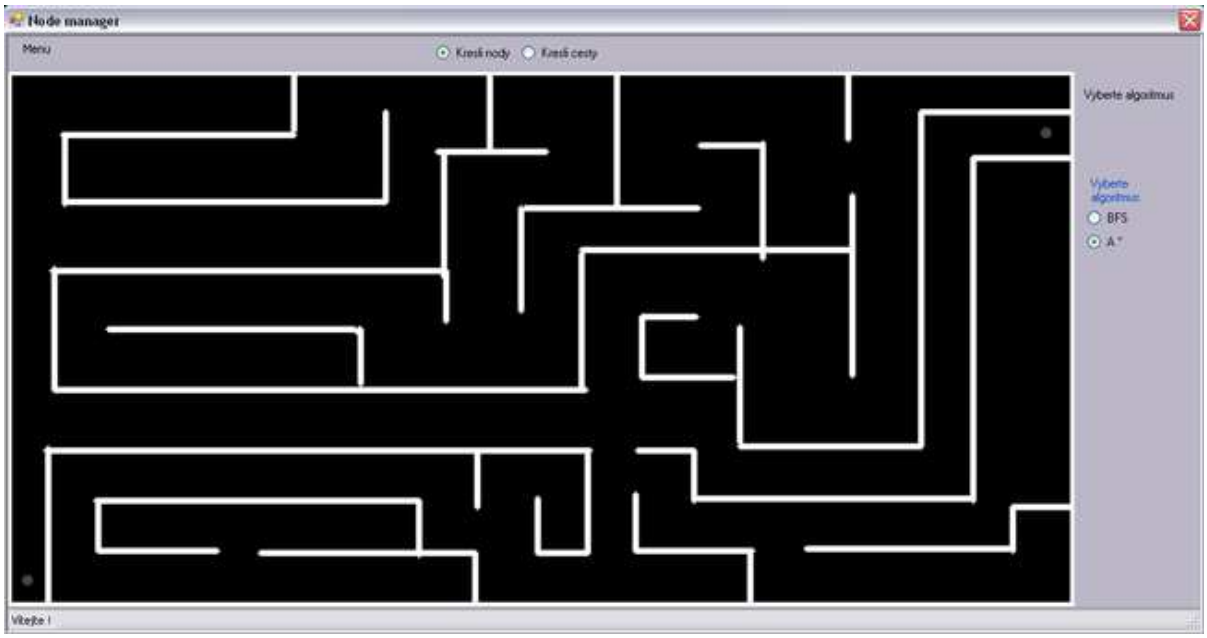
Seznam příloh

Příloha 1. Ukázka souboru s uloženými informacemi o terénu

```
-nodes
-48.33659 0.0 -20.09804
-47.84736 0.0 -7.843137
6.65362 0.0 -7.254902
16.34051 0.0 -8.333333
16.63405 0.0 -12.54902
38.16047 0.0 -12.94118
38.55186 0.0 18.62745
-ways
0 1
2 1
2 3
3 4
4 5
5 6
-info
path='E:\IBP\src\trunk\MyGame\teren\bludiste.raw'
width='512'
height='256'
water='0'
realW='100.0f'
realD='50.0f'
realH='20.0f'
alg='astar'
```

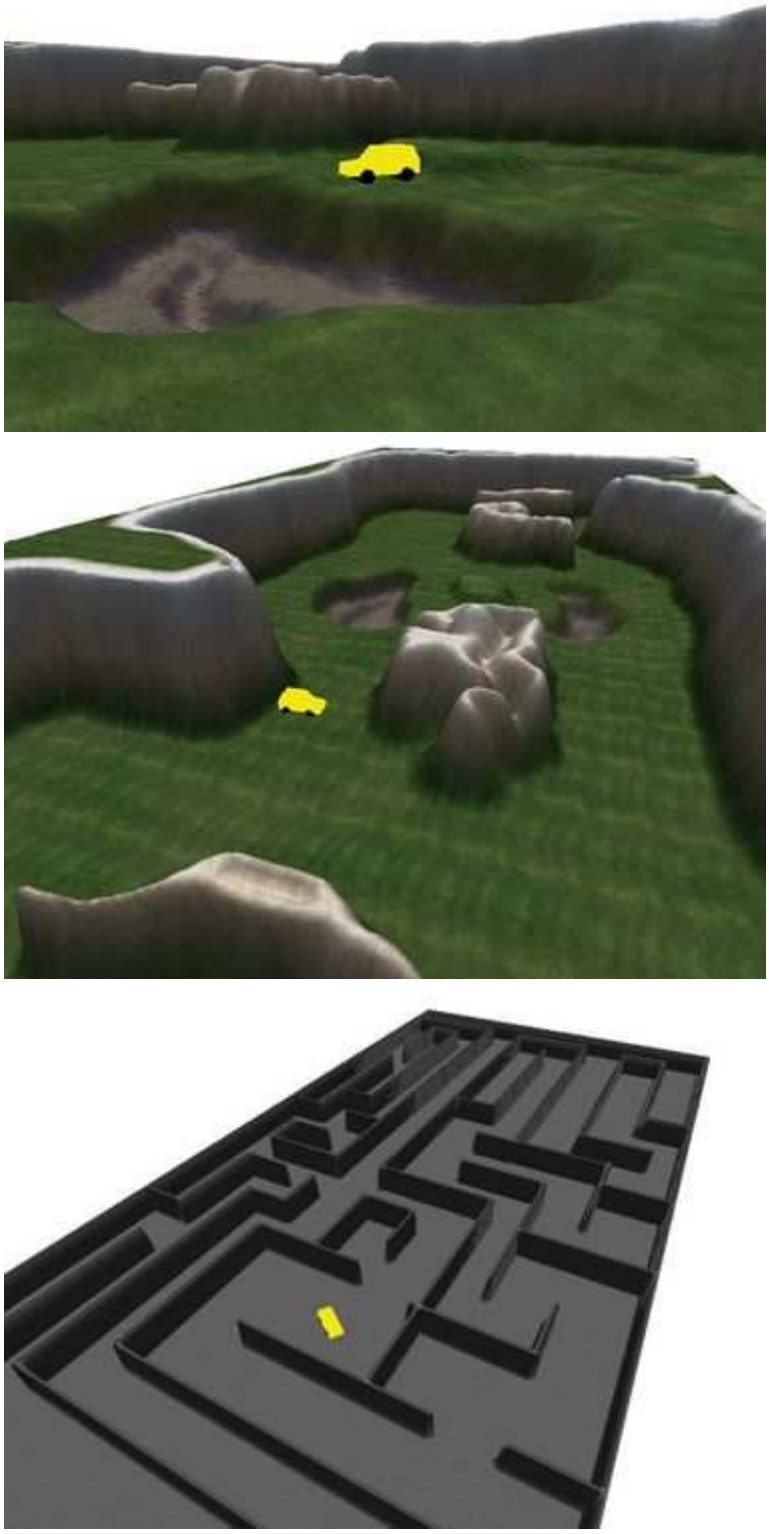
Takto vypadá soubor s konfiguračními údaji pro vykreslování terénu. V sekci –node jsou uloženy pozice uzlů. V sekci –ways jsou indexy uzlů určující odkud kam se lze pohybovat. Následuje cesta k výškové mapě, velikost výškové mapy a skutečná velikost terénu v metrech. Poslední položkou je zvolený algoritmus prohledávání.

Příloha 2. Vzhled terénu a vzhled výškových map



Ukázka výškové mapy bludiště.

Ukázka výškové mapy terénu.



Ukázka obrázků z demonstračního programu na vyhledávání nejkratší cesty.

Příloha 3. DVD

Přiložené DVD obsahuje následující adresáře a soubory:

- DirectX_SDK – obsahuje kompletní instalaci SDK directX 9 potřebnou ke kompilaci projektu MyGame
- DirectX_Redist – distribuce directX 9 potřebnou pro spouštění
- src – adresář obshuje veškeré zdrojové soubory
- bin – adresář obsahuje binární spustitelné soubory.
- readme.txt – soubor obsahující návod spouštění, instalaci programů